

SYNCLAVIER INTERNALS

(c) New England Digital Corporation

19-Jun-1987

The information contained in this manual is for internal use only.  
DO NOT send this out to offices or customers!

Section 1 -- Organization of Sources . . . . .	3
SYNSOU . . . . .	3
SYNRSOU . . . . .	4
OTHER CATALOGS . . . . .	6
Section 2 -- Internal Memory . . . . .	7
OVERALL STRUCTURE . . . . .	7
QUEUES . . . . .	8
NOTE BLOCKS . . . . .	9
PARTIAL BLOCKS . . . . .	9
Section 3 -- External Memory . . . . .	11
OVERALL STRUCTURE . . . . .	11
SEQUENCE BASE SECTORS . . . . .	14
TIMBRE PARAMETER AREA . . . . .	14
NOTE AREA HEADER . . . . .	15
TRACK HEADER SECTOR . . . . .	16
NOTE LIST SEGMENT . . . . .	17
NOTE RECORDS . . . . .	18
TIMBRE BANK . . . . .	21
MUSIC PRINTING SEQ INFO . . . . .	21
TIMBRE HEADS . . . . .	21
TERMINAL DISPLAY INF . . . . .	22
MONO SAMPLING BUFFERS . . . . .	23
LOOK UP TABLES . . . . .	23
Section 4 -- Poly Memory . . . . .	25
OVERALL STRUCTURE . . . . .	25
POLY HEADER . . . . .	26
20 SECTORS . . . . .	27
SOUND FILE BLOCKS . . . . .	27
Section 5 -- Hardware Overview . . . . .	29
DEVICES AND THEIR ADDRESSES . . . . .	30
DIRECT TO DISK . . . . .	31
Appendix . . . . .	32
TERMINOLOGY . . . . .	32
XPL TRICKS . . . . .	34
NOTE NAMES . . . . .	35

## Section 1 -- Organization of Sources

The Synclavier sources are kept in a tree of sub-catalogs. There are two primary trees:

### SYNSOU

Contains Synclavier sources which have been modularized, especially including code for the top-level, screen manager, sample-to-memory and utilities.

### SYNRSOU

Contains Synclavier real-time sources which have not yet been modularized.

### SYNSOU

The SYNSOU catalog contains many sub-catalogs which in turn contain the XPL source files. The currently defined sub-catalogs are:

- \* D40MOD - Printer output and hard copy
- \* D42MOD - Special output routines for DEC voice box
- \* DEBUGMOD - Sequence debugger code (for internal use only)
- \* DIRMOD - Synclavier directory screens
- \* EDITMOD - Sequence editor screen
- \* ERRMOD - Synclavier error messages
- \* FILEMOD - File searching code
- \* GETMOD - Code for get.next.event
- \* GLOBMOD - Global variables needed everywhere
- \* GPRMMOD - A set of general buffered disk read routines
- \* LINKMOD - Miscellaneous routines
- \* LODMOD - Routines to talk to the direct-to-disk over SCSI
- \* MATHMOD - High precision math and time conversion routines
- \* MNOTMOD - Music notation screen
- \* MONOMOD - Routines for mono sampling playback
- \* MOUSEMOD - Module for mouse code
- \* OPTMOD - Optical disk screen
- \* PARMOD - Module for processing Synclavier parameters
- \* PATCHMOD - Sound file patch screen - connects sound files to keys
- \* PLOTMOD - Plot library
- \* PMEMMOD - Old (16 bit) poly memory interface, now calls polymod
- \* POLYMOD - Module with code to access poly memory
- \* RECMOD - Recorder screen (another sequence editor)
- \* ROUTMOD - MIDI and multi-channel routing display screens
- \* SCRNMOD - Screen drawing primitives (draw, box, etc...)
- \* SEQMOD - Sequence manipulation primitives to operate on data str.
- \* SMGRMOD - Module for screen manager

PAGE 4

MISSING

PAGE 5

MISSING

## OTHER CATALOGS

There are several other top-level catalogs which contain files needed to compile the Synclavier software:

- \* SYNRDCLS - External declarations for real-time modules (SYNRSOU)
- \* SYNAUXS - Auxiliary literals used by only some modules
- \* SYNLITS - Basic common literals for the Synclavier
- \* SYNMODS - External declarations for each module in SYNRSOU
- \* SYNMAINS - Main programs and do files for Synclavier compilations
- \* SYNCOMS - Compilation\*flag sources for different system versions

There are also several catalogs which contain relocatable binaries for the Synclavier software:

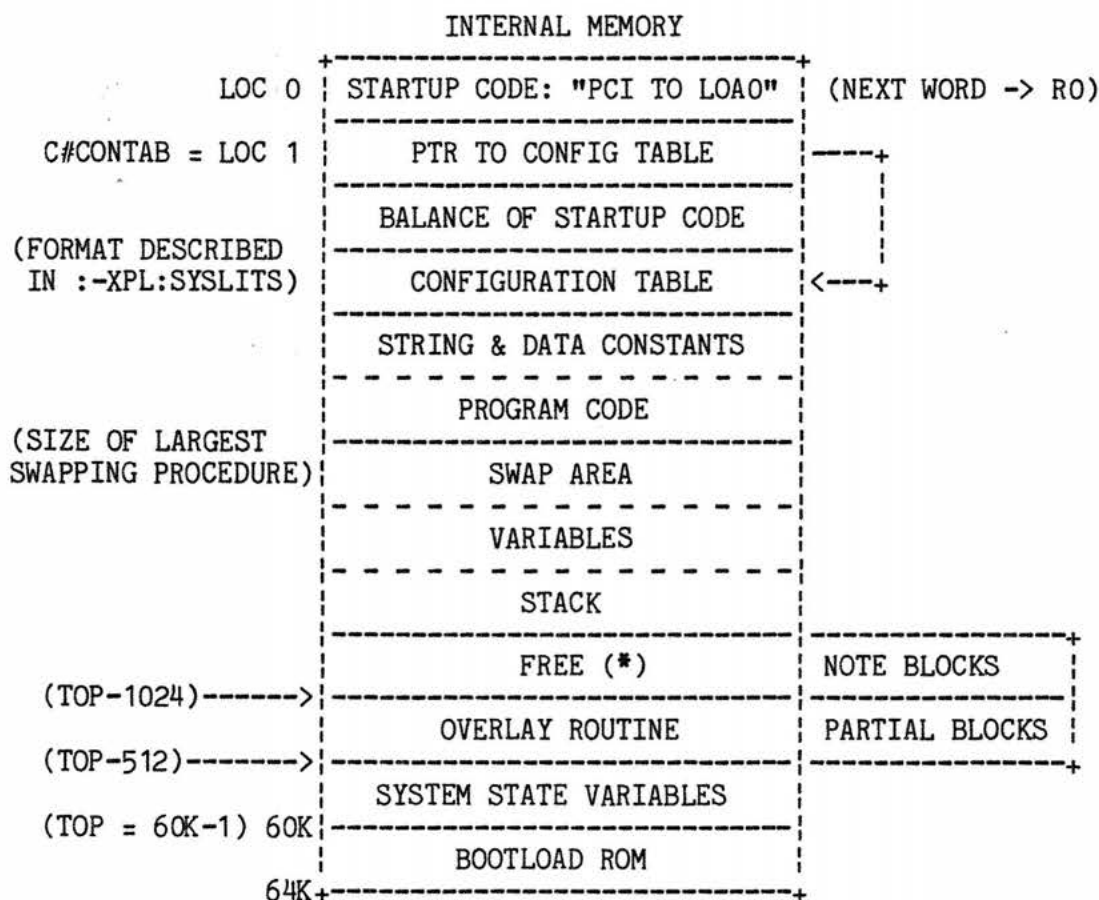
- \* SYNLIBS - Relocatable binaries for each module (from SYNRSOU)
- \* SYNTLIBS - Relocatable binaries for terminal screens (from SYNRSOU)
- \* SYNRLIBS - Relocatable binaries for Synclavier real-time modules (from SYNRSOU)

## Section 2 -- Internal Memory

There are several different kinds of memory in a Synclavier system. The ABLE computer has memory directly available to the processor known as INTERNAL MEMORY. There is additional memory used for storing sequences, timbres and other data structures known as EXTERNAL MEMORY. A system may also contain POLY MEMORY, used primarily for storing sampled sound data.

### OVERALL STRUCTURE

INTERNAL MEMORY is that which is directly available to the CPU for storing instructions and data. Information stored in INTERNAL MEMORY is accessible more quickly than other memory in the system. The 16 bit architecture of the ABLE processor limits the amount of INTERNAL MEMORY to 64K words. The structure of INTERNAL MEMORY looks like:



(\*) NOTE: WHEN RUNNING THE SYNCLAVIER PROGRAM, THIS SPACE FOLLOWING THE STACK IS USED FOR NOTE BLOCKS AND PARTIAL BLOCKS. THE "OVERLAY ROUTINE" IS SWAPPED OUT TO MAKE ADDITIONAL ROOM FOR THEM. THEY ARE ALLOCATED AT RUN-TIME OFF OF THE "HEAP" BY THE SYNCLAVIER PROGRAM.

If an XPL procedure is declared to be a swapping procedure it is not always resident in INTERNAL MEMORY. The compiler automatically takes care of allocating a place for it to be loaded in EXTERNAL MEMORY. When it is called for execution, it is copied for you to the "SWAP AREA" in INTERNAL MEMORY and program control is passed there. When another procedure is swapped in to the "SWAP AREA", the previous routine is over-written. Only the code is swapped in; the values of any static variables are preserved in INTERNAL MEMORY across swaps.

Most swapping procedures in the Synclavier code are declared to be "SWAPABLE" [sic]. This is a literal which translates to "RECURSIVE SWAP" for the XPL compiler. It means that variables default to AUTOMATIC (created on the stack) rather than STATIC (in internal memory).

The "SWAP AREA" reserved for a given program is the size of the single largest swapping procedure declared by that program. See the section in the XPL manual on "Swapping Procedures" for more information on the subject.

The Synclavier software is loaded into the PROGRAM CODE and SWAP AREA portions of INTERNAL MEMORY for execution. Like any executing program, it also uses the STACK and VARIABLES segments. QUEUES of pending notes are built in the NOTE BLOCK and PARTIAL BLOCK areas which are described in more detail below.

### QUEUES

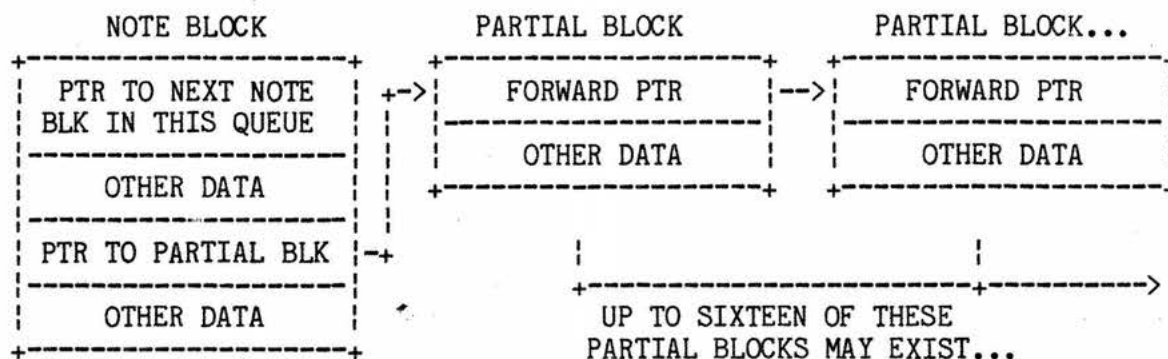
NOTE and PARTIAL BLOCK structures for notes which are currently active, or will be very soon, need to be in INTERNAL MEMORY for the sake of real-time speed of data access. Most of the sequence and timbre information is stored in EXTERNAL MEMORY, because it is much larger than INTERNAL MEMORY. The Synclavier software looks ahead 80msec in real time and builds QUEUES of NOTE BLOCKS which in turn point to zero to sixteen PARTIAL BLOCKS in INTERNAL MEMORY.

There are several QUEUES present in INTERNAL MEMORY. Each one has a global pointer to the start of the QUEUE which is a linked list of NOTE BLOCKS. The currently defined pointers and their QUEUES are:

- \* NOTELIST -> First NOTE BLOCK in the ACTIVE NOTE LIST QUEUE
- \* KBDLIST -> First NOTE BLOCK in the NEW KEYBOARD NOTES QUEUE
- \* SEQLIST -> First NOTE BLOCK in the NEW SEQUENCE NOTES QUEUE
- \* SEQLAST -> Last NOTE BLOCK in the NEW SEQUENCE NOTES QUEUE
- \* TIELIST -> First NOTE BLOCK in the TIED NOTES QUEUE
- \* TIELAST -> Last NOTE BLOCK in the TIED NOTES QUEUE
- \* NFREEP -> First NOTE BLOCK in the FREE NOTE BLOCK LIST
- \* PFREE -> First PARTIAL BLOCK in the FREE PARTIAL BLOCK LIST

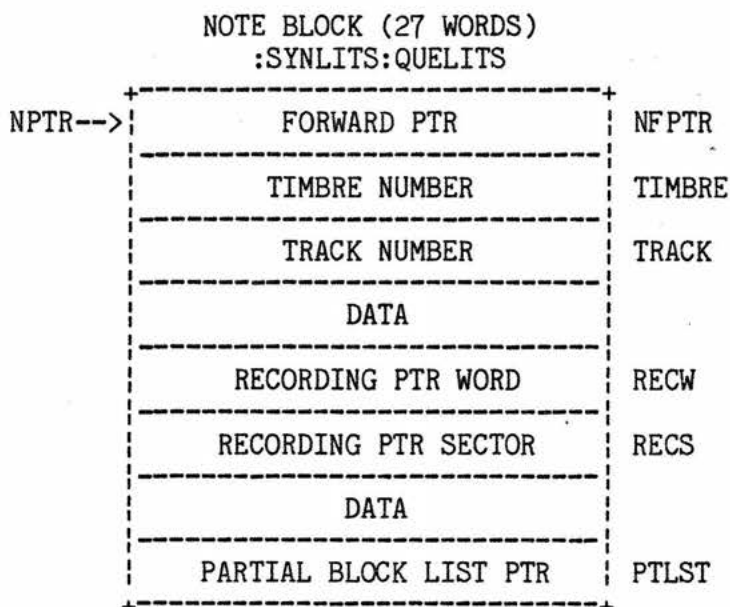


## QUEUE STRUCTURE



## NOTE BLOCKS

NOTE BLOCKS are allocated and placed on one of the QUEUES just described. In order to use the offsets defined for a NOTE BLOCK (eg: NFPTR, TIMBRE...), you must set the pointer NPTR to the start of the NOTE BLOCK you wish to access.



## PARTIAL BLOCKS

A chain of zero to sixteen PARTIAL BLOCKS are allocated and pointed to by the NOTE BLOCK just described. A note may have one to four partials by itself. This expands to a maximum of sixteen possible PARTIAL BLOCKS when you take the features "chorus" and "partial chorus" into consideration. In order to use the offsets defined for a PARTIAL BLOCK (eg: PFPTR, PLOG...), you must set the pointer PPTR to the start of the PARTIAL BLOCK you wish to access.

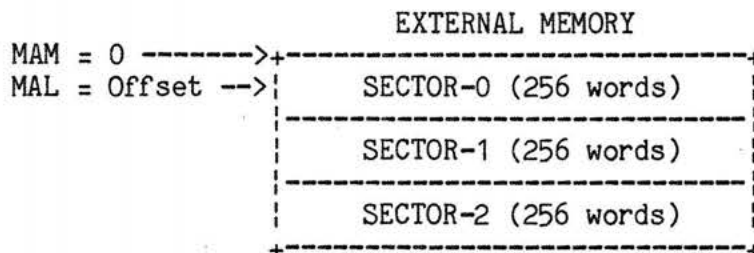
PARTIAL BLOCK (80 WORDS)  
:SYNLITS:QUELITS

PPTR-->	FORWARD PTR	PF PTR
	CHORUS PTR	PLOG
	DATA	
	VOLUME ENVELOPE Q FORW PTR	EQF
	VOLUME ENVELOPE Q BACK PTR	EQB
	FM ENVELOPE Q FORWARD PTR	IQF
	FM ENVELOPE Q BACKWARD PTR	IQB
	DECAY FORWARD PTR	FQF
	DECAY BACKWARD PTR	FQB
	DATA	
	GHOST BLOCK PTR	GF PTR

### Section 3 -- External Memory

External memory is not directly addressable by the ABLE CPU. It is accessed as a device is through the use of XPL "read" and "write" statements. Because it is addressed using two registers totalling 24 bits, it is possible to have a maximum of 32 megabytes of external memory.

External memory is divided into SECTORS of 256 words each. Two address and two data registers are used to access it. An address register "MAM" gives the sector number (Memory Address Most-significant). A second address register "MAL" is an offset into the selected sector (Memory Address Least-significant). Once an address is specified in "MAM" and "MAL", data (a 16-bit word) is read or written from the "MD" register. A fourth register "MDI" may be used to access data and auto-increment the MAM and MAL pointers to the next sequential word, crossing sector boundaries if necessary.



The values of MAM, MAL, MD and MDI are defined in :SYNLITS:GLOBLITS. The following example reads the first 1024 words of external memory into an array in internal memory, and leaves zeros in place of the data in external memory:

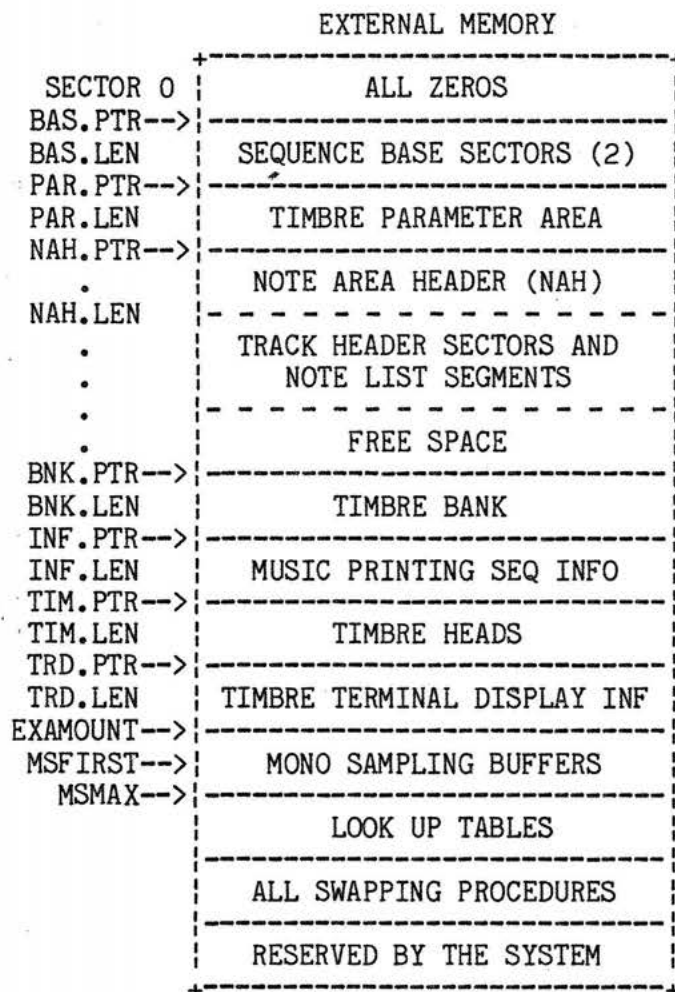
```
write(MAM) = 0;          /* Point at the first sector */
write(MAL) = 0;          /* No offset into the first sector */
do i=0 to 1023;
    array(i) = read(MD);  /* Read data word */
    write(MDI) = 0;       /* Clear location and point at next */
end;
```

#### OVERALL STRUCTURE

When the Synclavier program is running, the lower addresses of external memory are used to store sequence, timbre and note information for the user. Swapping procedures are stored in the upper addresses. Swapping procedures are allocated at compile time; all other structures shown here are built at run-time.

The major data structures begin at a pointer, named ???PTR with a length of ???LEN. Note that the .LEN value is the current length of real data stored in that particular area. There is often a "gap" maintained at the end of each data structure so new entries can be

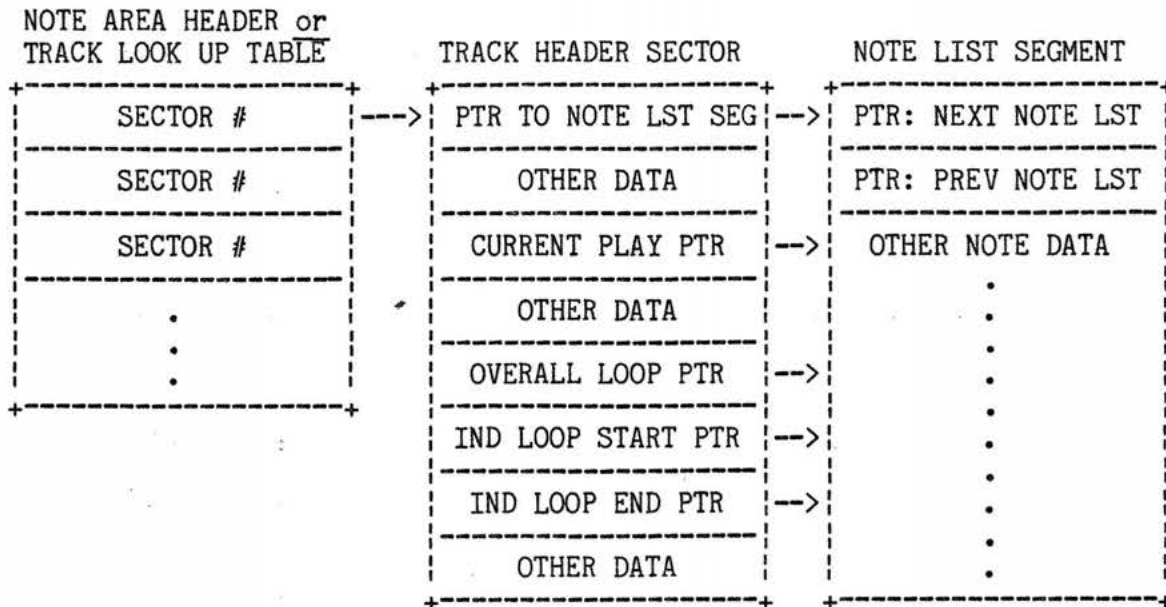
made without shuffling everything around. Usually the largest "gap" is kept following the NAH information to allow the sequence to grow quickly. As space becomes tight, the gaps are reduced - to zero if necessary.



The next diagram shows the relationship between the track header structures. The NOTE AREA HEADER is a single sector containing up to 256 pointers to TRACK HEADER SECTORS (relative to NAH.PTR). There is also a TRACK LOOK UP TABLE in the LOOK UP TABLES area which has the same structure, but is 256 absolute pointers to the TRACK HEADER SECTORS.

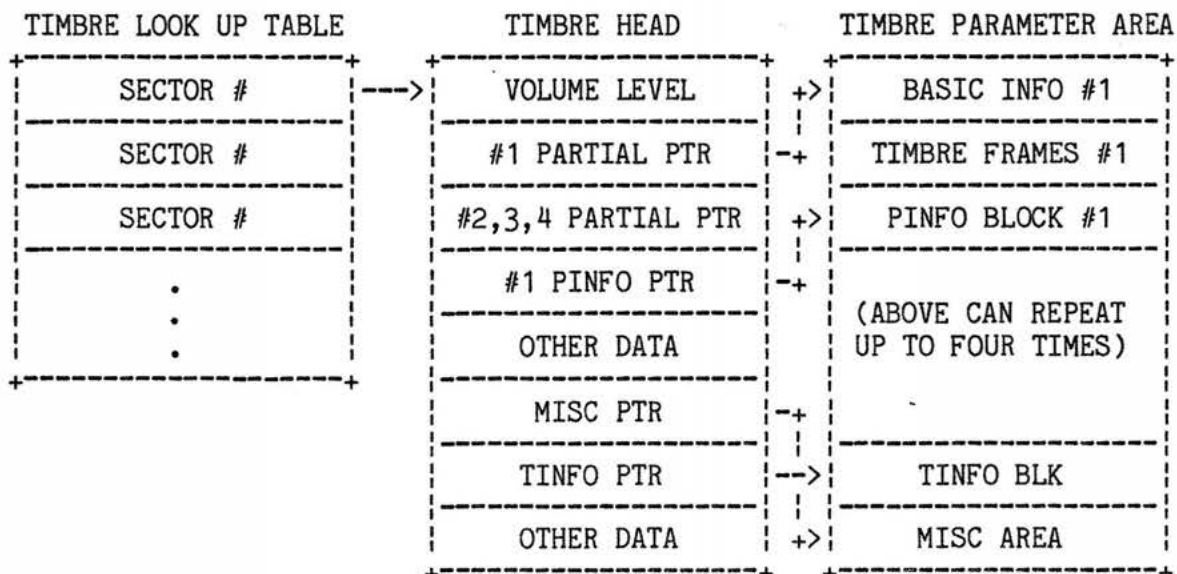
Each TRACK HEADER SECTOR has several pointers into the NOTE LIST SEGMENT for that track. They include pointers to the start of the NOTE LIST SEGMENT, to the currently playing note, and to the start and end of loops. The individual structures are described in more detail later.

## TRACK HEADER STRUCTURE



The next diagram shows the relationship between the timbre data structures. The TIMBRE LOOK UP TABLE is a single sector in the LOOK UP TABLES area containing up to 256 absolute pointers to TIMBRE HEADS. Each TIMBRE HEAD has several pointers into the TIMBRE PARAMETER AREA for that timbre. The individual structures are described in more detail later.

## TIMBRE STRUCTURE

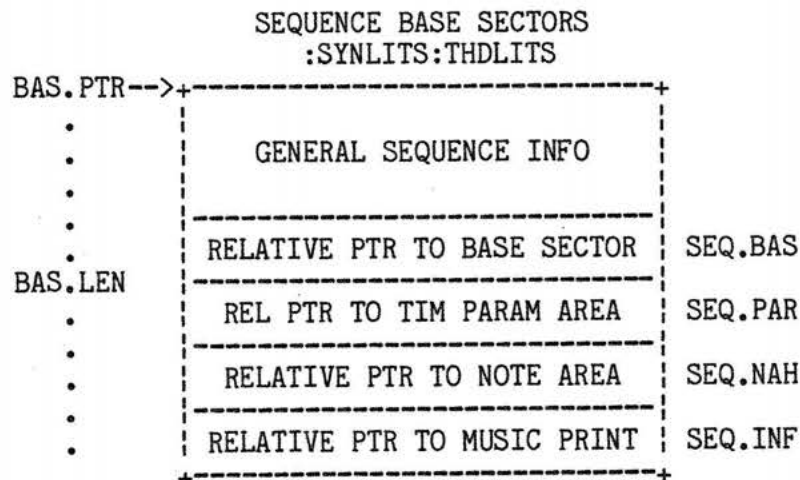


The following diagrams are an expansion of the previous general pictures of external memory data structures. No attempt is made to include every piece of data. Rather, the general types of information and their relative locations are shown. Pointers to other data structures are included so the inter-relationships can be seen.

It is worth noting that many of the pointers used in external memory are RELATIVE to a base pointer for the data structure containing them. This is so that information in external memory can be shuffled around without having to re-calculate all the pointers. Only absolute pointers become invalid.

### SEQUENCE BASE SECTORS

SEQUENCE BASE SECTORS contain miscellaneous information and pointers to timbres, notes and music printing. This information and the relative pointers are only updated when about to write a sequence out to the disk. They are not kept up to date during run time.



### TIMBRE PARAMETER AREA

The TIMBRE PARAMETER AREA is where the partials are stored for a timbre. A timbre can have up to four partials. Each partial starts out with a 46 word block (or just a word "SUP" (-2) if the partial is not used). Following the 46 word block are some optional blocks of data which are variable length. All four partials must be specified, or marked unused with "SUP". A good concise example of some code which steps through this structure may be found in the procedure "FIND.TIMBRE" in the file :SYNRSOU:03-PRIM:131-SDF1.

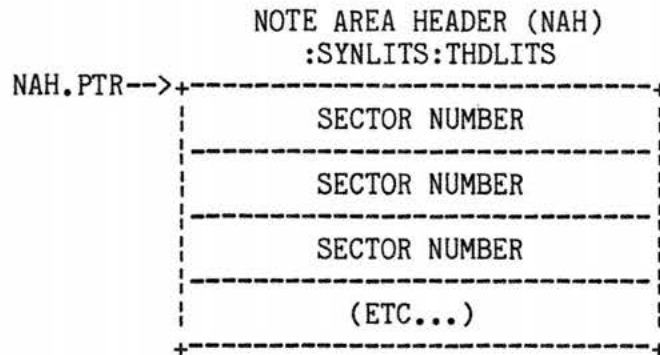
TIMBRE PARAMETER AREA  
:SYNLITS:PRMLITS

PAR.PTR-->	+-----+	
46=NUM.PARAMS	BASIC PARTIAL INFO (46 WDS) OR A SINGLE WORD "SUP" (-2)	P.*
OPTIONAL{	"MOR" (-1)	
{	-----	
TYPE{	"TF.TYPE" (0)	
{	-----	
CLEN{	"LEN.TIMB.FRAME" (172)	
{	-----	
{	TIMBRE FRAME (172 WORDS)	NOTE: THERE CAN BE MORE THAN ONE TIMBRE FRAME OR PATCH TIMBRE FRAME IN A PARTIAL.
{	- 8 SYSTEM WORDS	
{	- 12 PARAMETERS	
{	- 24 COEFFICIENTS	
{	- 128 WAVE TABLE ENTRIES	
OPTIONAL{	"MOR" (-1)	
{	-----	
TYPE{	"PT.TYPE" (1)	
{	-----	
CLEN{	"PT.LEN" (48)	
{	-----	
{	PATCH TIMBRE FRAME (48 WDS)	
{	-----	
OPTIONAL{	"PINFO" (-3)	
PINFO.LEN{	PARTIAL INFO (32 WORDS)	PI.*
	+-----+	
	NOTE: THE ABOVE REPEATS 4 TIMES (ONCE PER PARTIAL) AND THEN IS FOLLOWED BY:	
OPTIONAL{	"TINFO" (-4)	
TINFO.LEN{	TINFO BLOCK (96 WORDS)	TI.*
	-----	
	"MISC AREA" (8 WORDS)	
	+-----+	

NOTE AREA HEADER

The NOTE AREA HEADER is a single sector containing 256 pointers, one for each possible track. Each pointer is a sector number, relative to NAH.PTR, pointing to the track header sector for that track. If there is no active track corresponding to a given pointer, it is zero. The first two tracks are reserved for the keyboard and split keyboard respectively.





### TRACK HEADER SECTOR

One TRACK HEADER SECTOR exists for each of the active tracks. It is pointed to by the NOTE AREA HEADER data structure. Each TRACK HEADER SECTOR is exactly one disk sector in size. The sector contains miscellaneous information for the track such as loop and play pointers and real-time effects. There is also a pointer to the first NOTE LIST SEGMENT which is the first block of actual notes on the track.

There is a linked list of tracks, sorted by the next event to occur in real time. The head of this list is a global pointer, "NEXT.EVENT.QUEUE" which points to the TRACK HEADER SECTOR with the next up-coming event. Within each TRACK HEADER SECTOR are two pointers, THD.NEVF and THD.NEVR which are forward and reverse pointers to the next TRACK HEADER SECTORS in this list. As time advances, new TRACK HEADER SECTORS are appended to the end of the linked list. As an event occurs, the top track is removed from the head of the list.



TRACK HEADER SECTOR  
:SYNLITS:THDLITS

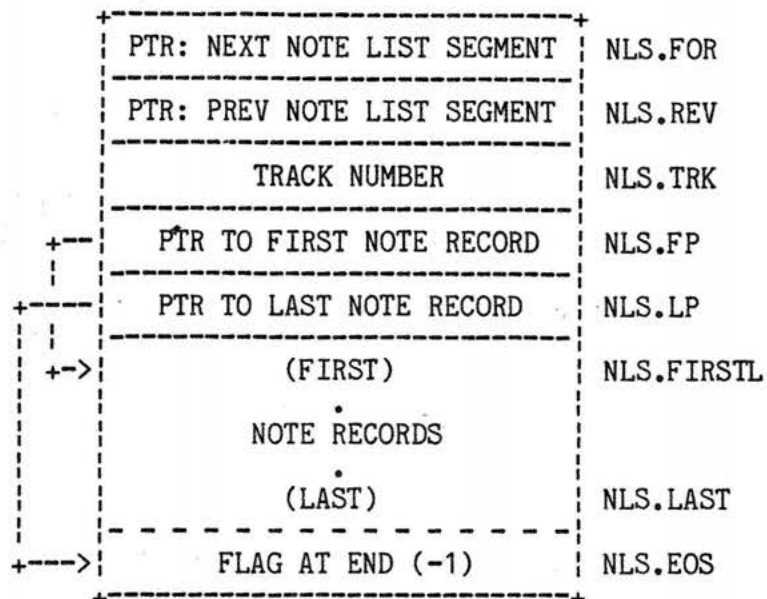
PTR TO START OF NOTE LIST	THD.FOR (*)
REVERSE PTR - ALWAYS ZERO	THD.REV
TRACK NUMBER	THD.TRK
FORWARD PTR TO NEXT EVENT	THD.NEVF
REVERSE PTR TO NEXT EVENT	THD.NEVR
BLOCK OF DATA FOR THIS TRACK	
CURRENT PLAY PTR - WORD #	THD.WRD (*)
CURRENT PLAY PTR - SECTOR #	THD.SEC (*)
MORE DATA FOR THIS TRACK	
OVERALL LOOP PTR - WORD #	THD.LP.WRD (*)
LOOP PLAY PTR - SECTOR #	THD.LP.SEC (*)
MORE DATA FOR THIS TRACK	
INDEPENDENT LOOP PTR - W #	THD.ILS.WRD (*)
INDEPENDENT LOOP PTR - S #	THD.ILS.SEC (*)
MORE DATA FOR THIS TRACK	
IND LOOP END PTR - WORD #	THD.ILE.WRD (*)
IND LOOP END PTR - SECTOR #	THD.ILE.SEC (*)
MORE DATA FOR THIS TRACK	

(\*) INDICATES A POINTER INTO THE NOTE  
RECORD LIST FOR THIS TRACK

NOTE LIST SEGMENT

A NOTE LIST SEGMENT is a linked list of sectors containing note records. Each sector contains a forward and reverse pointer and up to 126 2-word note records. Special format note records may also exist which are 4 words long. Note that the pointers to the first and last note records point at actual data. NLS.FIRST and NLS.LAST define the size of the block which can POTENTIALLY hold note records.

NOTE LIST SEGMENT  
:SYNLITS:THDLITS



NOTE RECORDS

NOTE RECORDS contain information about notes to play, including starting time, duration and key to play. They are also used to make changes in real-time effects which can effect some or all of the currently active notes.

There are three general rules used in determining the format of a NOTE RECORD:

1. If the LSB of the first word is set, the NOTE RECORD length will be 4 words, else it will be 2 words.
2. If the MSB of the first word is set, the NOTE RECORD is in "alternate format", if clear it is in "normal format". When in alternate format, the next 4 upper bits of the first word are used to indicate what type of "alternate" record it is.
3. For all 4 word records, the lower 6 bits of the last word must be "62" (VEL.NOTE). This is necessary to determine the length of the record when stepping through in reverse order.

A NORMAL FORMAT 2 WORD RECORD is the simplest NOTE RECORD. It represents notes in the range of 12 to 72 (C1 to C6). The velocity is assumed to be full value (RTE.MAX = 255).

NOTE RECORD (NORMAL FORMAT, 2 WORDS)  
:SYNLITS:SYNCLITS

DATA PRESENT-->	0 HI-DUR START TIME DELTA 0
BITS IN FIELD-->	1 ..4... .....10..... 1.
	LOW DURATION   KEY NUMB
	.....10..... .....6.....

A NORMAL FORMAT 4 WORD RECORD is used to represent keyboard notes 0 to 84 (C0 to C7). The full range of note velocity values may also be stored in this record. The duration and starting time fields are the same.

NOTE RECORD (NORMAL FORMAT, 4 WORDS)  
:SYNLITS:SYNCLITS

DATA PRESENT-->	0 HI-DUR START TIME DELTA 1
BITS IN FIELD-->	1 ..4... .....10..... 1
	LOW DURATION   KEY NUMB
	.....10..... .....6.....
	SWITCH/FLAG BITS   RTE DATA
	.....8..... .....8.....
	VOLUME/OTHER   RAISE/LOW   62
	.....8..... .....2..... .6.

An EXTENDED REST note record is needed to specify a rest of longer than 1023 milliseconds in a single note record.

NOTE RECORD (ALT FORMAT: EXTENDED REST, 4 WORDS)  
:SYNLITS:SYNCLITS

DATA PRESENT-->	1  0  START TIME DELTA 1
BITS IN FIELD-->	1 ..4... .....10..... 1.
	MOST SIG START DELTA
	.....16.....
	LEAST SIG START DELTA
	.....16.....
	UNUSED FIELD   62
	.....10..... .6.

An INDEPENDENT LOOP START record is used to mark the start of a loop present in this track independent of other tracks. Once a track has entered a loop, it will continue playing between the start and end loop records without exiting the loop. Only the first word of the 4 word record contains any useful data.

NOTE RECORD (ALT FORMAT: INDEPENDENT LOOP START, 4 WORDS)  
:SYNLITS:SYNCLITS

DATA PRESENT-->	1	1	LOOP START TIME DLT	1
BITS IN FIELD-->	1	.4.	.....10.....	1.
-----				
UNUSED FIELD				
.....16.....				
-----				
UNUSED FIELD				
.....16.....				
-----				
UNUSED FIELD				62
.....10.....				.6.
-----				

An INDEPENDENT LOOP END record marks the end of valid note records for this track. As time continues to advance, the track will loop indefinitely back to the INDEPENDENT LOOP START record.

NOTE RECORD (ALT FORMAT: INDEPENDENT LOOP END, 4 WORDS)  
:SYNLITS:SYNCLITS

DATA PRESENT-->	1	2	LOOP END TIME DELTA	1
BITS IN FIELD-->	1	.4.	.....10.....	1.
-----				
UNUSED FIELD				
.....16.....				
-----				
UNUSED FIELD				
.....16.....				
-----				
UNUSED FIELD				62
.....10.....				.6.
-----				

For more information on the exact format of the data in each of the above fields, refer to the definitions in :SYNLITS:SYNCLITS or to Tim Schaaff's memo "SYNCLAVIER NOTE RECORD FORMATS".

### TIMBRE BANK

A TIMBRE BANK is a temporary storage area used to hold a bank of timbres read in from the disk. The format of the TIMBRE BANK is the same as the TIMBRE PARAMETER AREA (see earlier description above). When the user selects a timbre bank, all the timbres in it are copied to this area. Then when a specific timbre is picked, it is copied to the TIMBRE PARAMETER AREA.

If the entire bank will not fit in external memory, the system waits for the user to select a specific timbre. Then, that one timbre is copied directly from disk to the TIMBRE PARAMETER AREA. The size of the TIMBRE BANK is often zero, either because no new partials are being selected, or because the space was needed for other things (eg: active sequence and partial information).

### MUSIC PRINTING SEQ INFO

The MUSIC PRINTING SEQUENCE INFORMATION is a block of data which is defined by and maintained for the music printing software. The format is considered "unknown" by the Synclavier software. A pointer to this block is maintained (INF.PTR) and the data is shuffled around to accommodate other areas as necessary.

### TIMBRE HEADS

The TIMBRE HEADS area is made up of sectors containing precomputed information for the basic timbre and pointers to partial data in the TIMBRE PARAMETER AREA. All of the pointers into the TIMBRE PARAMETER AREA are relative to PAR.PTR.

The TIMBRE HEADS area may also contain KEYBOARD LOOK UP TABLES. These are used when different sounds are mapped to various keys on the keyboard (eg: bass drum on middle-C, snare drum on middle-D, etc...); this is known as "patching". Each of the four partials for a timbre may be patched through a KEYBOARD LOOK UP TABLE, or they may not. The pointers in the TIMBRE HEAD are zero if no KEYBOARD LOOK UP TABLE exists for a partial; if non-zero they are relative to TIM.PTR. In the Synclavier code, sometimes the KEYBOARD LOOK UP TABLES are also referred to as the KEYBOARD PATCH LIST.

TIMBRE HEAD :SYNLITS:TIMLITS		
TIM.PTR-->+	VOLUME LEVEL	TIM.VOLUME.LEV
	PARTIAL PTRS (4 WORDS)	TIM.PARTIAL.POINTERS
	PINFO PTRS (4 WORDS)	TIM.PINFO.POINTERS
	SYNTH TYPE (4 WORDS)	TIM.SYNTH.TYP
	KEYBOARD LOOKUP PTRS (4 WDS)	TIM.KBDTAB.PTR
	PRE-COMPUTED DATA	
	MISCELLANEOUS AREA PTR	TIM.MISC.POINTER
	TINFO PTR	TIM.TINFO.POINTER
	PRE-COMPUTED DATA	
	FREE SPACE (BAL OF 1 SECTOR)	

KEYBOARD LOOK UP TABLE  
(ONE SECTOR; NO LITERALS DEFINED)

PTR TO PATCH TIMBRE FRAME IN TIMBRE PARAMETER AREA	} THIS REPEATS } THREE WORDS } PER KEY FROM } C0 TO C7:
0 IF MONO; 1 IF STEREO	} - 85 KEYS
PTR TO FILE BLOCK IN EXTERNAL OR POLY MEMORY	} - 255 WORDS } TOTAL
.	
.	
.	

TERMINAL DISPLAY INF

The TERMINAL DISPLAY INF area is scratch space used by the terminal for generating informational displays. An area is established to store data matching that which is displayed on the screen. Then, as time passes, information inside the Synclavier software is compared with the copy of the displayed data. If and when the data differs from what is on the screen, the terminal must be updated. This way, time only needs to be spent re-drawing parts of the display that are out of date. This technique is only one of many uses of the TRD area for displays.

### MONO SAMPLING BUFFERS

A MONO SAMPLING BUFFER is created in external memory to hold sound files when they are loaded from disk for the FM synthesizer. This block is only created on systems having FM synthesis. The format of the data is almost identical to sound files stored in poly memory. The MONO SAMPLING BUFFER is created with the smaller of the sizes:

- \* 158 Sectors.
- \* One half of all of external memory from the beginning up to the start of the LOOK UP TABLES.

### LOOK UP TABLES

Some pre-computed LOOK UP TABLES are stored at the top of external memory, just under the swapping procedures. These include LOG, SINE and FREQUENCY tables. This saves the time of calculating the data during real-time Synclavier execution. A number of other blocks of information are stored here. Although not expanded further in this document, each block is named, shown in its relative position in upper external memory and the size in sectors is shown. For more information, see the file :SYNRSOU:08-INIT:600-INIT containing the procedure "SETUP.EXTERNAL.ALLOCATIONS".

# LOOK UP TABLES

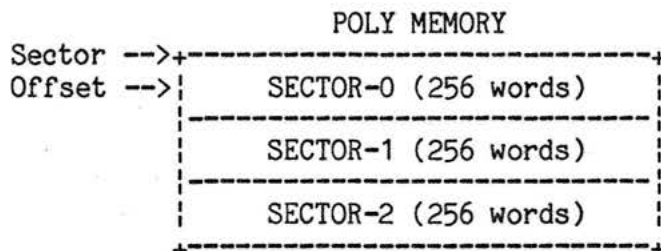
MFM.PTR-->	FM TO MULTI-CHAN MAP (1)	
MPOLY.PTR-->	POLY TO MULTI-CHAN MAP (1)	
USE.PTR-->	USAGE COUNTERS (1)	
CLAV.PTR-->	KEYBOARD BITS (1)	
TBUT.PTR-->	TRACK BUTTON TABLE (1)	
PBN.PTR-->	BUTTON LOOKUP TABLE (2)	
LOAD.PTR-->	SPACE TO HOLD LOADER (4)	
KBD.PTR-->	PRE-COMP KBD PTL BLKS (47)	
NUL.PTR-->	NULL TIMBRE (1)	
IMG.PTR-->	SCREEN IMAGE (4)	
VMAP.PTR-->	MIDI VELOCITY MAP (2)	
TTMAP.PTR-->	TRANSIT TIME MAP (2)	
FRE.PTR-->	FREQUENCY TABLE (8)	
STB.PTR-->	SINE TABLE (1)	
LTB.PTR-->	LOG TABLE (4)	
	KEYBOARD TIMBRE HEAD (1)	<--+
TIM.HEAD-->	TIMBRE LOOK UP TABLE (1)	--+ 1st entry
	KEYBOARD TRACK HEADER (1)	<--+
TRK.HEAD-->	TRACK LOOK UP TABLE (1)	--+ 1st entry



## Section 4 -- Poly Memory

Poly memory is not directly addressable by the ABLE CPU. It is accessed as a device is through the use of XPL "read" and "write" statements. Because it is addressed using two registers totalling 24 bits, it is possible to have a maximum of 32 megabytes of poly memory. This is in the process of being increased to 28 address bits allowing up to 512 megabytes of poly.

Poly memory is divided into SECTORS of 256 words each. When accessing this memory, the poly system must have the function code set to either read or write to poly memory. Then a single data register is used to specify the address and data. The data register must receive first the sector address, then the offset into the sector, and finally the data word(s). Poly memory accesses always cause an auto-increment after each one.



The values of all of the poly memory function codes are defined in :SYNLITS:FCODLITS. The following example reads the first 1024 words of poly memory into an array in internal memory. Note that the procedures PSMREAD(MSB,LSB); and PSMWRITE(MSB,LSB); are usually used by the Synclavier software to set up the function code and address information. This means the first three statements could be replaced with "psmread(0,0);".

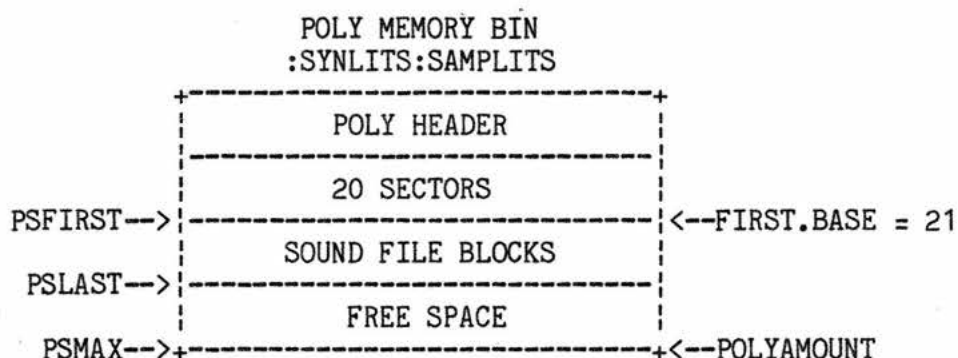
```
write(PSF) = PSRMA;      /* Set the poly system to read memory */
write(PSD) = 0;           /* Point at the first sector */
write(PSD) = 0;           /* No offset into the first sector */
do i=0 to 1023;
    array(i) = read(PSD); /* Read data word (and increment) */
end;
```

### OVERALL STRUCTURE

Poly memory is primarily used for storing sound files. It is possible to load sound information into poly memory, exit the Synclavier program, return later and have the sound files still present. They are usually only removed when the space is needed for other sound files, on command by the user, or when the system is powered down.

The system is in the process of being expanded to allow multiple poly bins to exist. Each one will have a maximum of 32 voices and 512 megabytes of memory (former maximum was a single bin with 32 voices and 32 megabytes of memory). Each bin has its own sound files loaded in it, and each bin is having data structured as shown below for one bin.

The first sector of poly memory is used to store a small header. The "MAGIC NUMBER" and "REVISION" at the start are zeroed out during sensitive transitions (eg: loading a new sound file; shuffling the existing sound files around) and restored when they are complete. This way, if a user manually aborts the Synclavier program while the data structures are in transition, everything must be restored from the disk.



#### POLY HEADER

The POLY HEADER is stored in the first sector of poly memory. It keeps a "MAGIC NUMBER" which means valid data is present if the number is preserved. There are also pointers to the other key data structures. These pointers are important since the sound file information is preserved in poly memory even when the Synclavier program is exited.

The two word pointers (MSB and LSB) are being created to support poly bins with greater than 32MB of memory. Older versions of the software use single word pointers for PSFIRST through PSHERE.

#### POLY HEADER

POLY.MAGIC (12345)	(30071 OCTAL)
POLY.REV (54335 REL-N)	(152077 OCTAL)
POLYAMOUNT: MSB,LSB	
PSFIRST: MSB,LSB	
PSLAST: MSB,LSB	
PSMAX : MSB,LSB	
PSFREE: MSB,LSB	
PSHERE: MSB,LSB	
(BALANCE OF SECTOR UNUSED)	

#### 20 SECTORS

Between the POLY HEADER and the SOUND FILE BLOCKS, there are 20 sectors reserved. This is a scratch area used to store a sound sample that is displayed on the terminal for in-depth analysis. A user can examine this sample in detail (eg: mousing through parts of it forward, backward, at varying speeds...).

#### SOUND FILE BLOCKS

The rest of poly memory is reserved for SOUND FILE BLOCKS. These are data areas of varying length containing the digitized sound samples which can be played back through the poly synthesizer. There is one "special" SOUND FILE BLOCK. If the first word of the filename is a "1", rather than a printable ASCII character, the block is the SOUND FILE CACHE. This is a block of pointers to where all the sound files are stored on disk.

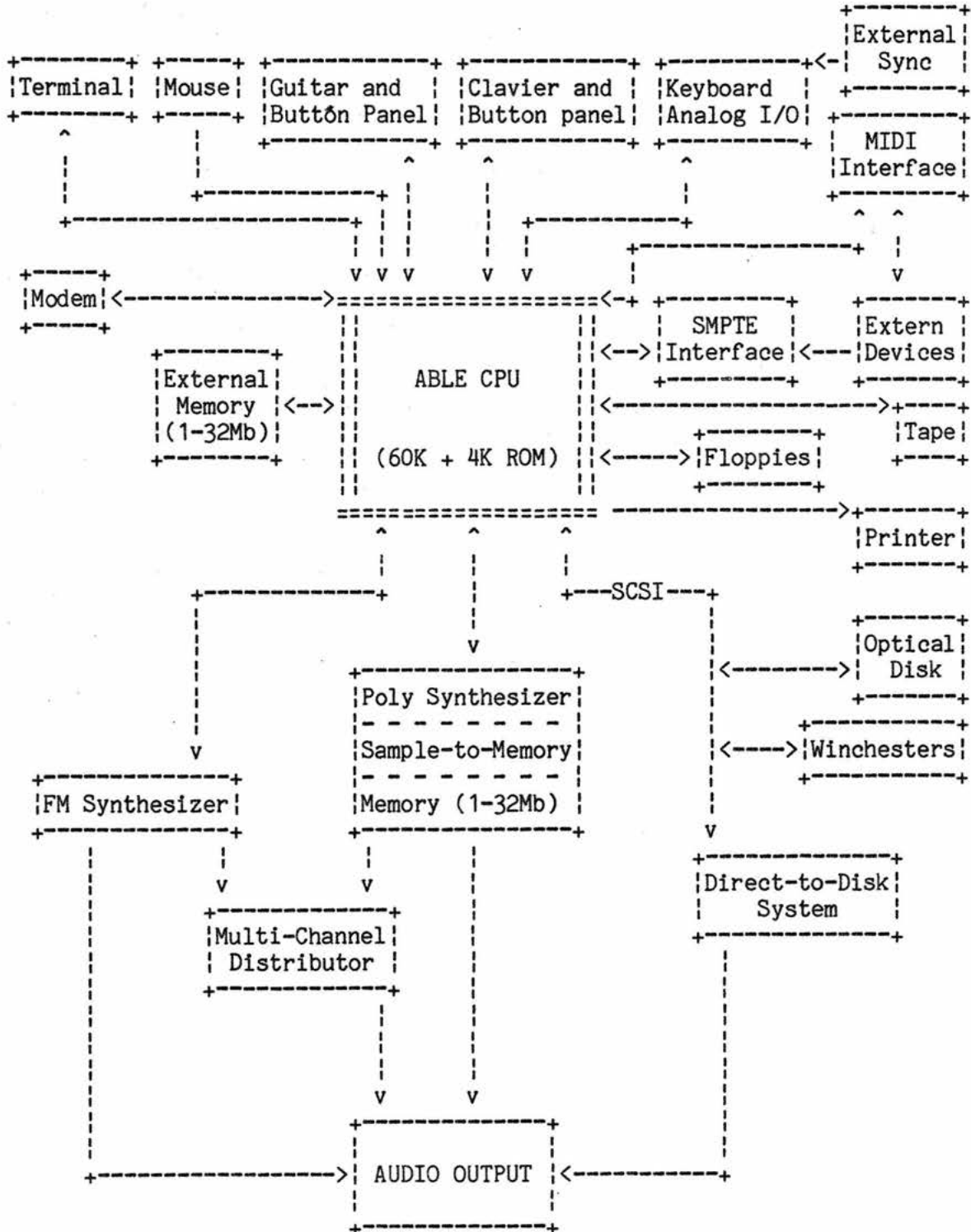
SOUND FILE BLOCK  
:SYNLITS:SAMPLITS

BLOCK LENGTH	BL.LEN = 0
NUMBER OF USERS	BL.USERS = 1
FILENAME - 4 WORDS	BL.FNAME = 2
FREE SPACE	6 to 19
BLK OF DATA MATCHING OFFSETS IN PATCH TIMBRE (PT.*)	BL.KEYTC = 20 to BL.SAVED = 43
FREE - REST OF 1ST SECTOR	44 to 255
ONE SECTOR OF FILE HEADER	SF.HDR = 1
TWO SECTORS OF SYMBOL TABLE	SF.SYM = 2
N SECTORS OF SAMPLE DATA	BL.POLY = 1024
.	
.	
.	

## Section 5 -- Hardware Overview

This is a block diagram of the Synclavier hardware system. The direct-to-disk system in the lower right is expanded later on.

The Synclavier Hardware



## DEVICES AND THEIR ADDRESSES

The following devices are connected to the ABLE buss to support the full configuration shown in the previous diagram:

D4567 - Multiply/divide (really D4,D5,D6,D7)

D3 - Real time clock (used to generate 5 msec interrupts)

D16 - Scientific timer (used to generate 1 msec interrupts)

D24 - SCSI host adaptor

D30TD - Kennedy 15mb cartridge tape drive

D32X - Buss extender card containing:

D32 - Multichannel distributor interface

D154-157 - Poly synthesizer interface

D34 - Guitar and button panel interface

D40Q - Combined serial port card containing:

D40 - Printer port

D42 - Modem port

D44 - Mouse port

D50 - Terminal port

D60 - External memory interface

D70 - Combined interface address containing:

D7-(8) - SMPTE interface

D70-(16) - MIDI interface

D100A - Floppies

D130 - Clavier and button panel interface (velocity keyboard)

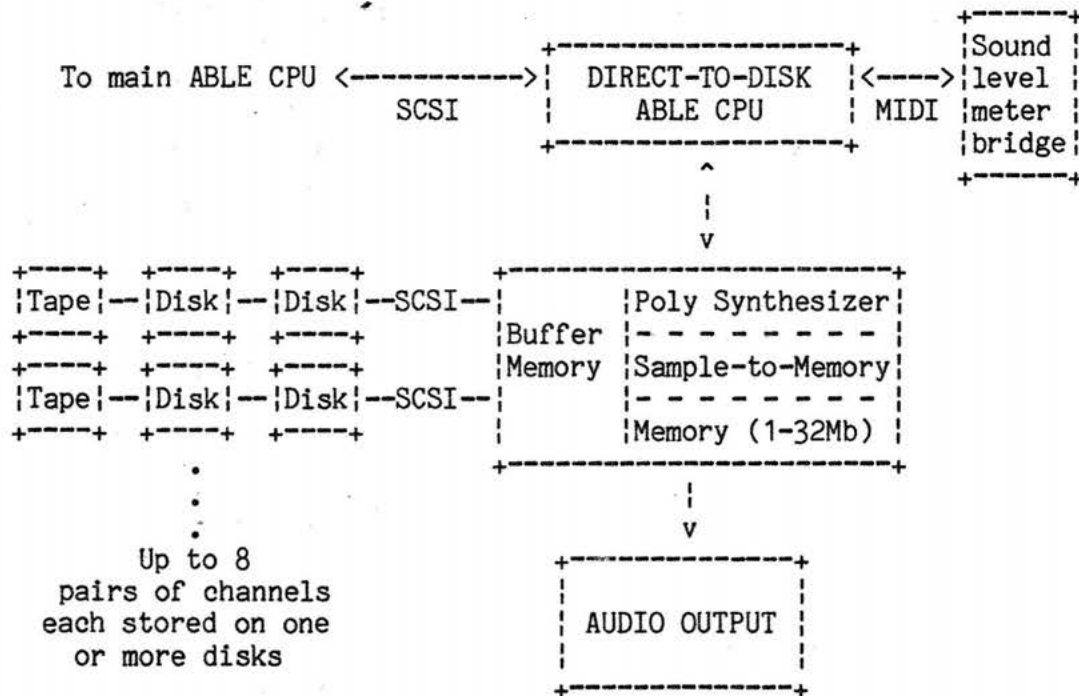
D160 - FM synthesizer interface

D160 - Keyboard analog I/O interface (original keyboard only)

## DIRECT TO DISK

The direct-to-disk system connects to the main ABLE computer via the SCSI buss. It contains another ABLE computer which is the same as the main one except it does not boot from a floppy. The boot ROM is modified to receive data from the main ABLE over the SCSI; there are no floppies or winchester disks on the "Direct-to-disk ABLE".

The Direct-to-Disk System



## Appendix

### TERMINOLOGY

This is a partial list of terms one is likely to encounter when reading through the Synclavier code.

CHANNEL - A voice allocated in a synthesizer. There must be a channel available for each individual sound which occurs simultaneously.

EXTERNAL MEMORY - Memory which is not immediately addressable by the ABLE CPU. It is accessed in sectors using 24 address bits and therefore may be up to 32 megabytes in size. Sequences, timbres and other data is stored here.

GHOST BLOCK - If a partial block is needed and none are available, a search is made to see if there are some decaying notes. A GHOST BLOCK is a partial block that is still in use, but is for a note which is decaying and will not be needed for long. Often, it is better to cut a decaying note short rather than avoiding starting a new note on time.

INTERNAL MEMORY - Memory which is directly available to the ABLE CPU for instructions and data. Due to the 16-bit address space in the processor, it is limited to 64K words.

LIVE CLICK TRACK - A "metronome" recorded on a track, rather than using the one supplied by the Synclavier. This allows for changes in tempo during a song.

LOOPS - An overall loop may be established to cause all the tracks to loop between marked "start" and "end" points. An "independent" loop may be set up for a single track.

MP INFO - Music printing info area; the "INF" section of external memory.

NAH - NOTE AREA HEADER; a data structure in external memory.

NLS - NOTE LIST SEGMENT; a data structure in external memory.

ORK - The "ORiginal Keyboard" for the Synclavier II. It is the predecessor of the "Velocity Keyboard". It has 61 notes (C1 to C6).

PATCH - When a sound files are divided so that different keys or ranges of keys on the keyboard generate sound from different samples.



PINFO AREA - An optional data area in the TIMBRE PARAMETER AREA in external memory. It contains extra partial information needed in some cases.

POLY MEMORY - Memory which is part of the poly synthesizer, used primarily for storing sampled sound data. This memory is also addressed using 24 bits and may be up to 32 megabytes long. This is in the process of being increased to 28 bits and 512 megabytes.

RIBBON - An effect controller located above the keyboard on some systems. As you slide your finger back and forth on the ribbon, an associated sound parameter can be modified.

RTE - Real-time effects; selectable from the button panel on the keyboard.

SECTOR - A block of memory or disk storage 256 words (512 bytes) in size. External memory, poly memory and disks are accessed in sectors and 8-bit offsets within sectors.

TIMBRE - A sound or set of sounds available to be played from the keyboard (or guitar). Typical timbres include drums, flute, trumpet, vibes... They are often produced from several samples of these instruments at different pitches and then "patched" together to cover the range of the keyboard. Or, in the case of percussion for example, different instruments may be on different keys. Timbres can also be totally synthesized (eg: a sine wave) rather than samples of sounds.

TINFO AREA - An optional data area in the TIMBRE PARAMETER AREA in external memory. It contains special real-time effects information.

USAGE KEY NUMBER - A checksum generated for a timbre which has a VERY VERY high probability of being unique (though not 100%). Used to identify a timbre to see if it is already loaded without having to compare it bit-for-bit.

VELOCITY KEYBOARD - The current Synclavier keyboard which is both velocity and pressure sensitive. It is often referred to as the "New Keyboard". It has 76 notes (A0 to C7).

WESTERN SCALE - The "standard" 12-tone scale: C, C#, D, D#, E ... A#, B.

VOICE - A channel allocated in a synthesizer. There must be a voice available for each individual sound which occurs simultaneously.

## XPL TRICKS

These are some of the more unusual constructs you will find when reading through the Synclavier code.

'core(0)' - This can be used as a literal definition, for example:

```
DCL OVERALL.VOLUME LIT 'CORE(0)';
```

This is sometimes done when a section of code containing a literal is conditioned out, but will not make it through the first passes of the compiler without SOME value. core(0) is not guaranteed to contain anything useful to most programs, so care must be taken not to actually use the value in real code.

FREE.R0; - This is a literal defined to be the (seemingly useless) code:

```
IF 0 THEN WRITE("300") = READ("300");
```

This statement is optimized out by the third pass of the compiler. Its purpose is to trick the XPL compiler into thinking that R0 ("300") has just been used. Then one can write some assembler code like "write("301") = a + b;" to put a value into R1 for some extra fast piece of code. Since the compiler always allocates registers using R0 first, you know the addition of a and b will be done in R0, and R1 will be free for your use.

REMINDERS - Sometimes you will see "dead" code like:

```
IF POLY.SPLICE THEN DO; END;
```

This is put in place as a reminder so that later you can find it to put real code in. By removing the literal definition of POLY.SPLICE compile errors will be generated if you forget to replace one of these constructs with real code.

RPT nnn; - An XPL literal construct used with the Model-C (or newer) processor only. It causes the next single word instruction to be executed nnn times. The execution takes place in micro-code as "one instruction cycle" for the processor so it is not interruptable! Care should be taken not to make these loops too large (usually limited to RPT 64;). It is FAST though.

SWAPABLE - A literal defined as "RECURSIVE SWAP". Most swapping procedures in the Synclavier source are declared with this pseudo-keyword. Note that swapping procedures may NOT be called during an interrupt.

REGISTER 13 - The code:

```
WRITE("313") = SOMETHING;
```

is storing a number directly in register 13. XPL never uses R13, so it is available as a very fast variable within a procedure.

## NOTE NAMES

When notes are stored in the Synclavier software, they are generally numbered from 0 to 84 with #36 as "middle-C". This is convenient since 85 keys is  $(3 * 255)$ . The keyboard look up table for example is three words per key and fits in one sector of external memory. It is also seven full octaves C-C. The keys are named as follows:

<u>NOTES</u>	<u>NUMBERED</u>
C0 - B0	0 - 11
C1 - B1	12 - 23
C2 - B2	24 - 35
C3 - B3	36 - 47
C4 - B4	48 - 59
C5 - B5	60 - 71
C6 - B6	72 - 83
C7	84

Sometimes other designations are used for notes and pitches. MIDI uses keys numbered from 1 to 127 where 60 is "middle-C".

"Pitch class numbers" use numbers of the form "O.SSCC" where "O" is the octave (corresponding to the octave numbers in the table above, eg: A3 is in octave 3), SS is the semi-tone (0 to 11) and CC is the number of cents (hundredths of a semi-tone). Using this notation, "3.0900" would be "A3" (A at 440 Hz); "3.0000" would be "C3" ("middle-C" or C at 523.3 Hz).

The original keyboard has a range of 61 notes from C1 to C6. The new velocity keyboard is 76 notes from A0 to C7.